NASA Applied Information Systems Research Program

ANNUAL REPORT: “Improving Swift”

PI: Harold F. Levison (SwRI)
CoI: Martin Duncan (Queen’s University)
CoI: Mark Lewis (Trinity University)
Programmer: David E. Kaufmann (SwRI)

1. PROJECT OVERVIEW

Swift is a software package that has been developed over the last ten years by the PI and CoI Duncan in order to perform high-quality, long-term numerical integrations of planetary systems. Due in part to its ease of use, its support by the PI, and its free availability to the planetary community, it has become an industry standard and is used all over the world. We proposed to make several fundamental improvements to Swift that would significantly enhance its utility to the planetary community.

The formation of the Solar System is one of the oldest unsolved problems in physics. And yet, the solution to this problem has crucial implications for issues concerning humanity’s place in the universe and the likelihood of other habitable planets existing within the Galaxy. Thus, understanding the origin and evolution of our Solar System and planetary systems around other stars is one of NASA’s fundamental goals.

By and large, the formation and evolution of the Solar System as a whole is dominated by the gravitational interaction of the bodies that constitute it. There is now overwhelming evidence that the dynamics of the Solar System is chaotic (in the mathematical sense). Thus, the most powerful method for understanding its formation and subsequent dynamical evolution is to construct numerical computer models that mimic the physical processes that are believed to have been important.

There are two basic ways in which numerical simulations can be used to help unravel the mysteries of Solar System formation and evolution. The first is to design numerical experiments whose goals are to construct and dynamically evolve planets themselves. Perhaps as useful, the structure of small body reservoirs (asteroids and comets) contain important clues to the final stages of planet formation since they were witnesses to and bear the scars of this period. Thus, much is to be learned by constructing models of the origin and evolution of these reservoirs as well.

The construction of both types of models present challenges that are unique to this field and require specific algorithms designed to handle them. These challenges include:

1. The Solar System is very old compared to the orbital periods of its constituent objects. In particular, the age of the Solar System is 4.5 billion years, while Mercury’s orbital period is only 88 days (0.24 years). Because even modern algorithms require at least 20 or 30 iterations to resolve an orbit, long-term simulations of the entire Solar System require hundreds of billions of serial iterations.

2. These simulations require that the gravitational forces between each pair of objects be calculated at every iteration. The currently available algorithms that are capable of accurately and efficiently performing billions of iterations require that the force between each pair of objects be calculated individually. Thus, if there are N bodies in a simulation, then there are $O(N^2)$ force calculations each iteration. The number of objects initially required to perform planet formation simulations can be very large indeed, making each iteration computationally expensive.

3. Objects grow in these simulations by collisional accumulation, or accretion. In addition, close gravitational scattering events play an important role in the overall dynamical evolution of these systems. These short-lived (~ 1 day at the Earth) events need to be accurately followed during the simulations, requiring much higher temporal resolution than the dominant orbital motions.

Performing a complete simulation of the origin and evolution of the Solar System is, and will continue to be for quite some time, beyond our ability. However, great strides toward understanding planet formation can be obtained through simulations of various pieces of the puzzle. Within the last decade, our ability to perform these simulations has been greatly increased by the development of algorithms that, due to their unique structures and energy conservation characteristics, allow for very long-term simulations. The most important three of these are:

1. the Wisdom-Holman Map (WHM),
2. the Regularized Mixed-Variable Symplectic (RMVS) algorithm, and
3. the Symplectic Massive Body Algorithm (SyMBA).

The ability of the planetary community to perform these types of simulations has also been significantly aided by Swift. The current version of Swift contains WHM and RMVS, as well as other well-known integrators. SyMBA has not yet been released as part of Swift because it is not yet stable enough, efficient enough, or user-friendly enough for general use. We proposed to make the following improvements to Swift:

1. Make SyMBA part of Swift and release it to the public. This task requires essentially all of Swift to be rewritten to incorporate SyMBA's distinct data structures, thereby allowing the other integrators to be made more efficient and the code to be made more user-friendly.
2. Employ more sophisticated methods for calculating the forces between objects. Hierarchical tree methods as well as fast multipole methods are being considered in this context. These methods perform the force calculation for N bodies faster than $O(N^2)$, typically $O(N \log N)$.
3. Parallelize the code to take advantage of the current popularity of computing clusters, thereby allowing the code to handle larger numbers of particles and thus be used for more realistic and more encompassing simulations.

2. PROGRESS IN YEAR 2

2.1. Swifter, the New Swift (Serial Version)

The rewrite of the serial (unparallelized) version of the new Swift (which we call Swifter) is largely complete. A grand total of approximately 180 units is currently anticipated for the Swifter package. Of this total, 173 (96%) have been written. The units still to be completed are the final pieces of SyMBA, the most complex of the integrators included in Swifter. New versions of the following integrators have already been completed:

1. the Wisdom-Holman Map (WHM),
2. the Regularized Mixed-Variable Symplectic (RMVS) algorithm,

3. the fourth-order $T + V$ Symplectic (TU4) method of Candy & Rozmus,
4. the Democratic Heliocentric (DH) method,
5. the fifteenth-order RADAU (RA15) integrator of Everhart, and
6. a Bulirsch-Stoer (BS) method.

The latter two integrators, while not symplectic in their nature, are included because they provide useful comparisons to the symplectic codes and they are still widely used in the field for certain problems.

2.2. SWIFTVis Visualization and Analysis Tool

CoI Lewis has written a Java-based data visualization and analysis tool called SWIFTVis for use with the Swifter package. This tool allows the user to construct graphically, execute, and save an analysis session by means of a full-featured GUI. The analysis session is built up by specifying and connecting input datasets, functions and filters, and output plots as a graph on a portion of the GUI. The remainder of the GUI is devoted to displaying the results to the user.

While the tool is tailored for use with the output from the Swifter programs, it is nevertheless flexible enough to allow the user to work with almost any type of data. Also, regardless of the input dataset type, SWIFTVis provides buffering techniques to allow the user to work with files larger than can fit into the available machine memory.

Within SWIFTVis, the bulk of the analysis is performed by means of functions and filters. These capabilities allow the user to specify and perform virtually any desired manipulation of the input data prior to visualization. For example, SWIFTVis currently provides the following filters: select, thinning, function, sort, merge, binned, and movie. There are currently plans to add a coordinate conversion filter and a resonance identification filter to this list.

SWIFTVis also provides the user the ability to visualize the analyzed data in various ways. For example, it supports scatter, streamline and surface plots, and allows multiple plots within a single plot window.

Finally, SWIFTVis has been written to be extensible. Each of the major components that can be added to the analysis graph (datasets, filters, and the data representations for plotting) follow a particular interface. Users can write their own implementations of those interfaces. In this way users can seamlessly add their own functionality to SWIFTVis.

2.3. The Parallelization of Swifter

We have recently ordered the Beowulf cluster intended to serve as the platform on which the parallelized version of Swifter will be developed. This cluster was purchased using internal Southwest Research Institute (SwRI) funds (not grant money) on the basis of a contingent capital equipment agreement. The cluster will consist of 25 compute nodes: 1 2.2GHz quad AMD Opteron machine and 24 2.133GHz dual AMD Athlon machines. The use of a cluster of multiprocessor machines will enable us to explore parallelization on both shared and distributed memory architectures.

At SwRI in Boulder we currently have access to older dual AMD Athlon machines. On these machines we have tested shared memory parallelization on prototype codes that isolate the portions of Swifter that we will seek to parallelize in the production code. Specifically, these are the routines that compute the interbody forces and determine which bodies are suffering a close encounter. In these runs we have seen performance increases of almost a factor of two. This gives us confidence that the production Swifter code will parallelize well.

3. WORK PLAN FOR YEAR 3

The following items constitute the work plan for the third year of the grant:

1. We will wrap up the serial (unparallelized) version of Swifter and distribute it to the planetary community. As part of this release we will provide extensive *User's* and *Programmer's Guides*, written in HTML, for Swifter.

2. After the Beowulf cluster arrives we will begin development of the parallelized version of Swifter. We will distribute this version as well upon its successful completion.

3. We will continue work on methods of achieving more efficient force calculations. We will develop the symplectic tree algorithm as well as investigate the use of fast multipole methods.

4. We will complete development of the SWIFTVis analysis tool and make it, together with all necessary documentation, available as part of the Swifter package.